Student Name: **Student id:** **Sect:** **#:**

QUESTION #	1	2	3	4	TOTAL
MAX POINTS	16	15	16	15	62
POINTS EARNED					

University of Bahrain College of Information Technology

Department of Computer Science

ITCS 332: Organization of Programming Languages FIRST TEST Date: MAR 23, 2015

****

QUESTION ONE:

- Fill in blanks **[4 pts]**

1) In denotational semantics, the state changes are defined by **MATHEMATICAL FUNCTIONS**.
   In operational semantics, the state changes are defined by **CODED ALGORITHMS.**

2) Give two suggestions to remove the ambiguity from grammars of arithmetic expressions:
   add operator precedence rules and **add operator associativity rules.**

3) An attribute grammar is a context-free grammar (BNF) plus 3 additions, name any two of them:
   SEMANTIC FUNCTIONS, PREDICATE FUNCTIONS, ATTRIBUTE VALUES.

4) The number of lexemes in a C++ statement: **if(!(23*ct/17)>99)dt=98** is **16** and the number tokens of is **10**.

- Convert each of the following BNF rules into ONE equivalent EBNF rule **[6 pts]**

5) <tfs> → <var> <G> <const> | <var> <G> <S> <const>
   <G> → @= | &= | #= | $=
   <S> → + | -

<tfs> → <var> (@=|&=|#=|$=) [(+|-)] <const>

6) <anyU> → <X> | <X> @ <anyU>| <X> ? <anyU>

<anyU> → <X> { (@ | ?) <X> }

- Convert each of the following EBNF rule into equivalent BNF rule(s). **[6 pts]**

7) <E> → <T> { (OR | XOR) <T>}
   <T> → <F> { (AND | NAND) <F>}

<E> → <T> | <T> OR <E> | <T> XOR <E>
<T> → <F> | <F> AND <T> | <F> NAND <T>

8) <G> → [(# | *)] <VAR>

<G> → <VAR> | # <VAR> | * <VAR>

QUESTION TWO: Carefully study the following grammar and answer the next 2 questions. **[15 pts]**

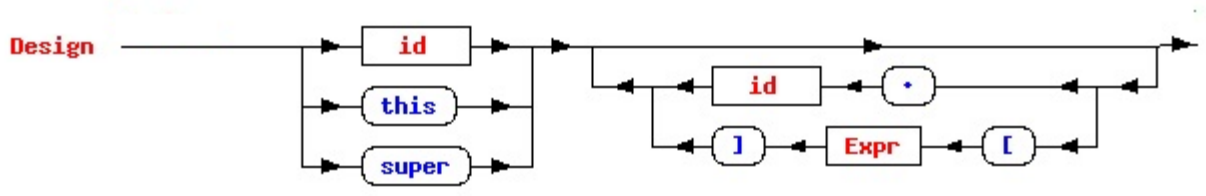
1) *<real>* → *<sign> <int> "#"| <sign> <int> "**E**" <sign> <int> "#"*
   *| <sign> <int> "." <int> "#"| <sign> <int> "." <int> "**E**" <int> "#"*
2) *<int>* → *<dig> | <int> <dig>*
3) *<sign>* → *"+" | "–"*
4) *<dig>* → *"**0**"|"**1**"|"**2**"|"**3**"|"**4**"|"**5**"|"**6**"|"**7**"|"**8**"|"**9**"*

1) Using the above BNF rules, construct the left-most derivations for the sentence: ***+40.57E3#***

2) Using the above BNF rules, construct the parse tree for the sentence: ***+40.57E3#***

3) Construct the syntax graph for the following EBNF rule.

<design> → *(<id> | "this" | "super“) { "." <id> | " [" <expr> "]" }*



QUESTION THREE: **[16 pts]**

- An mystery constant <**mys_const**> in some language is defined as a sequence of one <**dec_d**> or more decimal digits **(0,1,...,9)** preceded by **0x** or **9x** and may be followed by **#** or **!**. Write ONE EBNF rule to define valid mystery constants. Accepted constants: 0x34; 9x29!; 9x29; 0x29#;….

<mys_const> → (0x | 9x) <dec-d> { <dec-d> } [(# | !)]

- A declaration list <decList> is a sequence of one or more declarations <decL>. Each declaration is terminated by question mark "?" and consists of a type <type> followed by colon ":" followed by one or more identifiers <id> separated by hash "#". The type may be: int, real, bool, char. Write EBNF rule(s) to define valid declaration lists.

<decList> → ***<decL> ? { <decL> ? }***

<decL> → ***<type> : <ids>***

<type> → ***int \| real \|bool \| char***

<ids> → ***<id> { # <id> }***

- Write a BNF grammar for the language of nonempty data files. A nonempty data file <*datafile*> consists of one or more records, where each record <*rec*> is one or more fields. Each field <*field*> is either integer <*int*> (one or more digits <*dig*>) or string <*str*> (one or more characters <*char*> enclosed in double quotes). Every record ends with **$**. Every field ends with **#**.

<datafile> → ***<rec> $ \| <rec> $ <datafile>***

<rec> → ***<field> # \| <field> # <rec>***

<field> → ***<int> \| " <chars> "***

<int> → ***<dig> \| <int> <dig>***

<chars> → ***<char> \| <chars> <char>***

QUESTION *FOUR*: *Fill in blanks Questions* **[15 pts]**

1) The main bottleneck of compiling systems is **CPU-main memory speed gap**, and the main bottleneck of interpretations systems is **statement decoding**.

2) The optimization step in compilers is used to **speed up the execution** and **to reduce the size of the generated code**.

3) The generation step of a compiler accepts as input **intermediate code** and produces as output **a machine code**.

4) Give two suggestions to reduce the execution times of programs in any language without affecting its reliability: **reduce degree of optimization** and **use macros instead of subprograms.**

5) Java has better reliability than C++. Give two reasons: **Java performs index range checking while C++ does not** and **Java has no pointers while C++ has**.

6) Programming languages are implemented using 3 methods: **compiling systems**, **pure interpretation systems**, and hybrid systems.

7) The advantages of using preprocessor macros are: **Programs with macros are more generic (flexible)** and **Programs with macros execute faster.**

8) The compiler parser accepts as input **a set of lexemes** and produces as output **a parse tree**.

9) The overall simplicity of a programming language is affected by 3 features: The set of language constructs, **operator overloading**, and **feature multiplicity**.

10) The **more orthogonal** the design of the language, the **fewer exceptions** the language rules require, which makes the language easier to learn, read, and understand.

11) The main advantage of compiling systems is **fast execution**, and the main advantage of interpretations systems is **easy implementation**.

12) List two language features that affect its reliability: **type checking** and **index range checking**.

13) Web applications are implemented using a collection of languages such as **markup**, **scripting**, and general purpose languages such as JAVA.

14) Given: `R = b /4 - 3 {R > 5 }`, then the weakest precondition is **{b > 32}**.

15) Given: `T = 4X /(x-4) {T > 8 }`, then the weakest precondition is **{8 > x > 4}**.

{ type checking, statement decoding, scripting language, exception handling, set of lexemes, parser, machine code, scanner, process, data, parse tree, intermediate code, compiling systems, operator overloading, feature multiplicity, markup language, index range checking , pure interpretation systems, fast execution of programs, CPU-main memory speed gap, easy implementation, operation semantics, axiomatic semantics }.